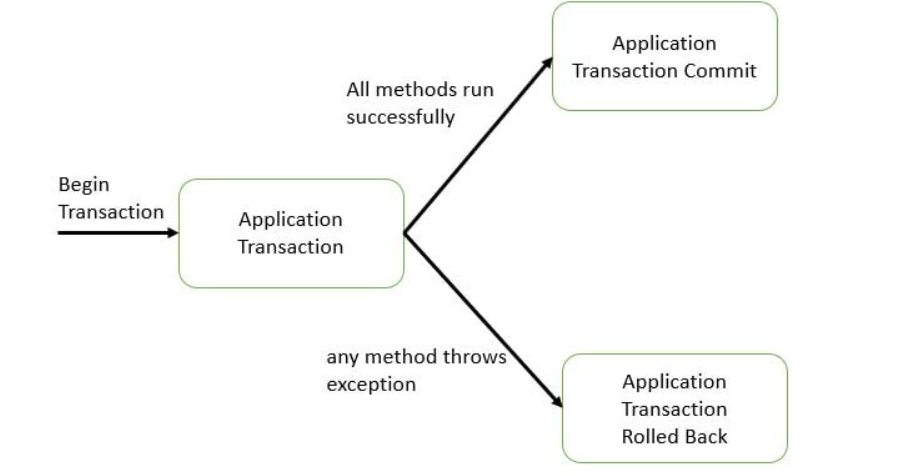
1. **Explain Spring Transaction Management?**

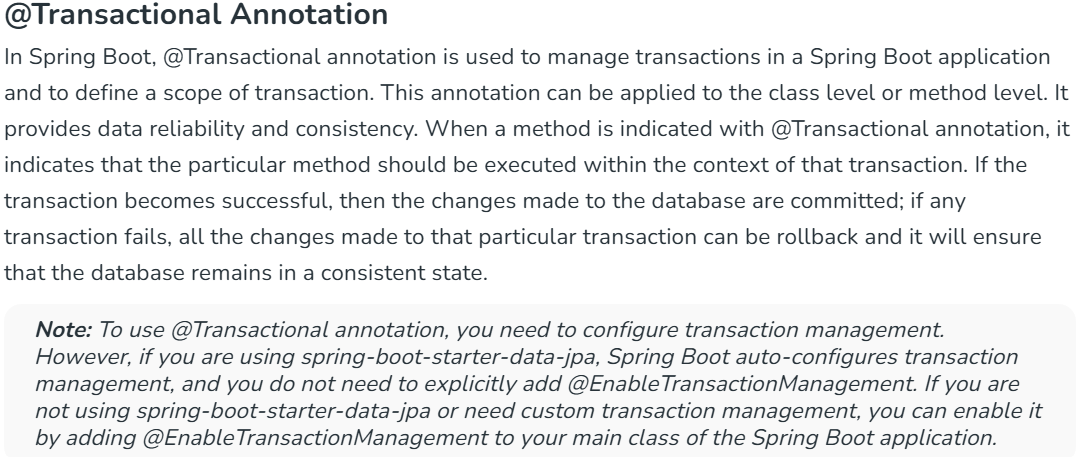
Spring provides powerful **transaction management** features that ensure data consistency and integrity in **multi-step operations** within applications. It handles transactions declaratively or programmatically, ensuring **ACID (Atomicity, Consistency, Isolation, Durability)** compliance in database operations.

Let's understand transactions with the above example. If a user has entered his information, the user's information gets stored in the user\_info table. Now, to book a ticket, he makes an online payment, and due to some reason(system failure), the payment has been canceled, so the ticket is not booked for him. But, the problem is that his information gets stored on the user\_info table. On a large scale, more than thousands of these things happen within a single day. So, it is not good practice to store a single action of the transaction(Here, only user info is stored, not the payment info).

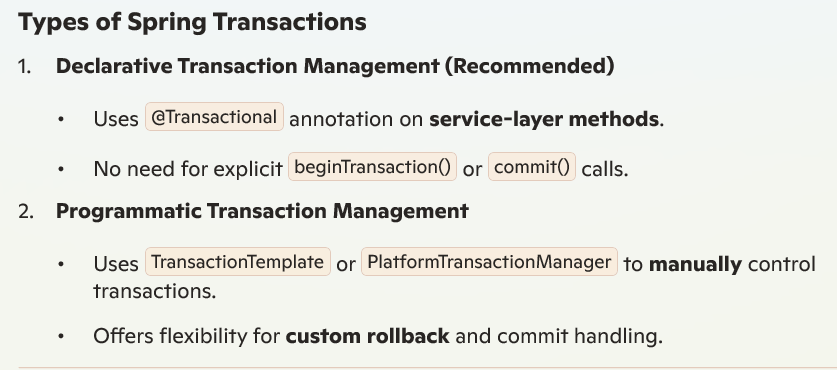
To overcome these problems, Spring provides transaction management, which uses annotation to handle these issues. In such a scenario, the spring stores the user information in temporary memory and then checks for payment information. If the payment is successful, then it will complete the transaction; otherwise, it will roll back the transaction, and the user information will not be stored in the database.

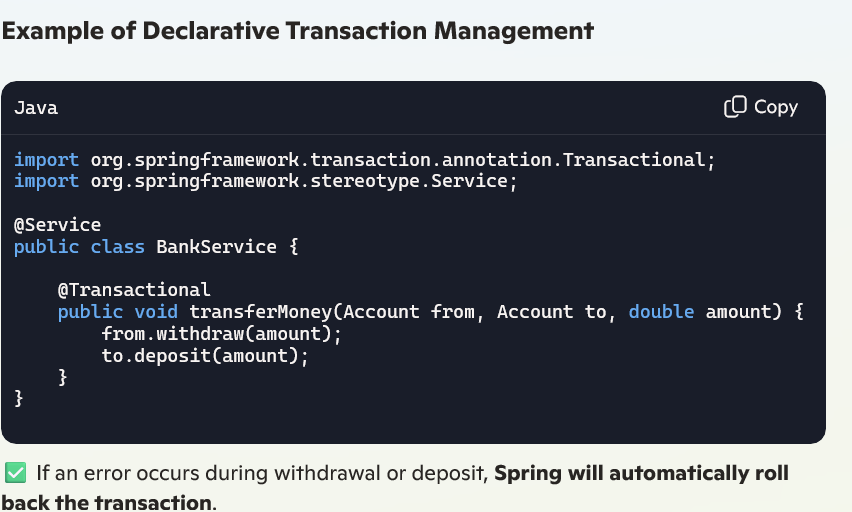


1. **How to handle the Spring Transaction Management?**



1. **Explain different types of Transaction management?**

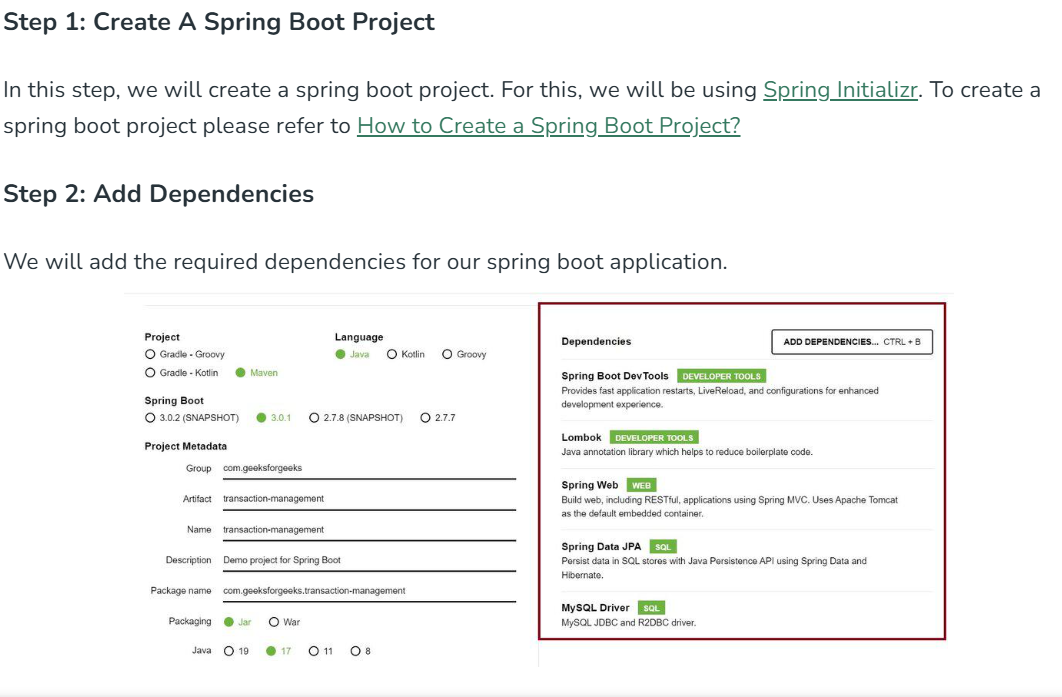


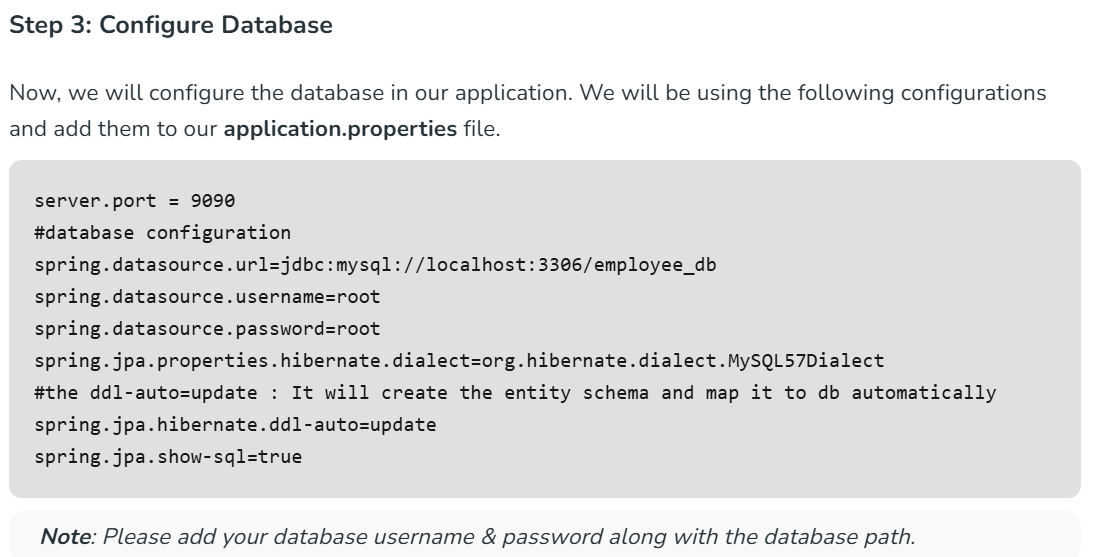


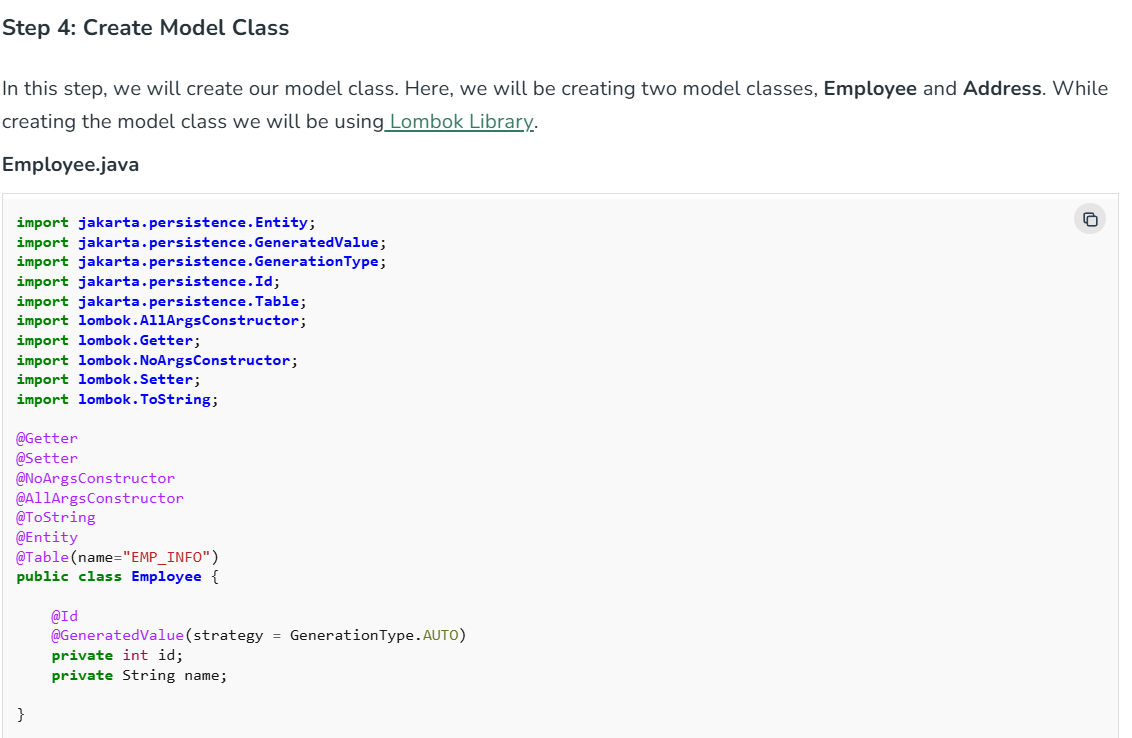




1. **Give an example for Spring Boot Declarative transaction management?**

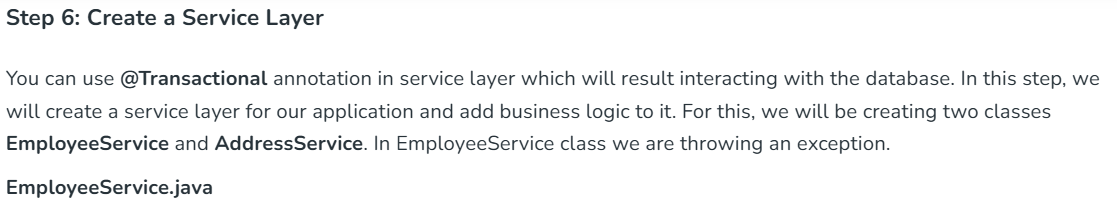






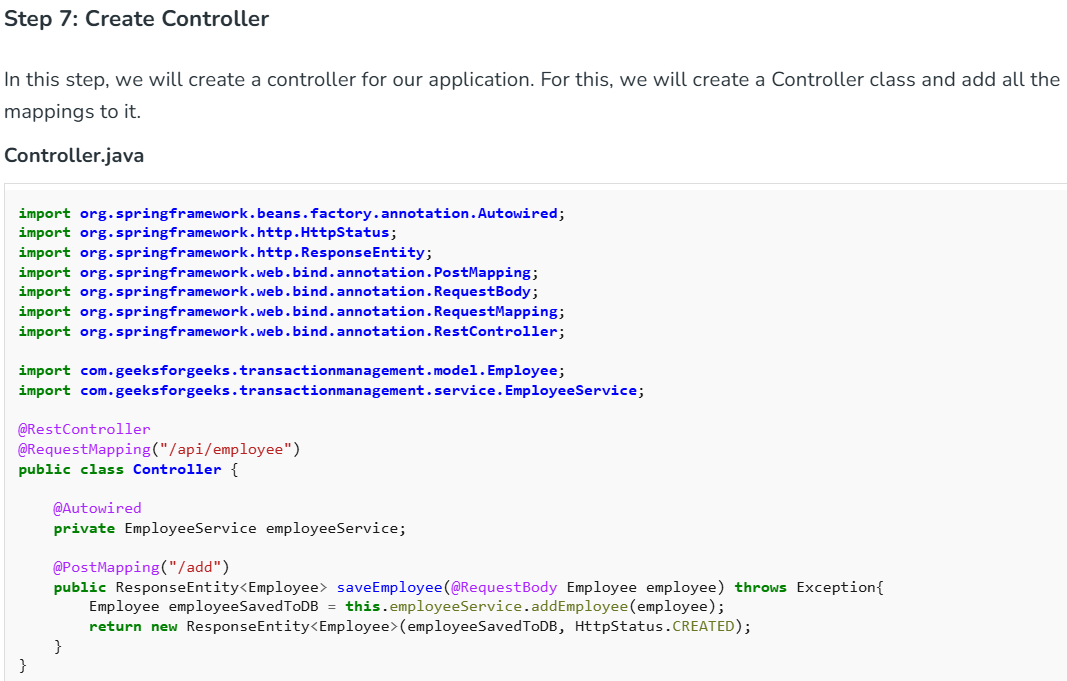


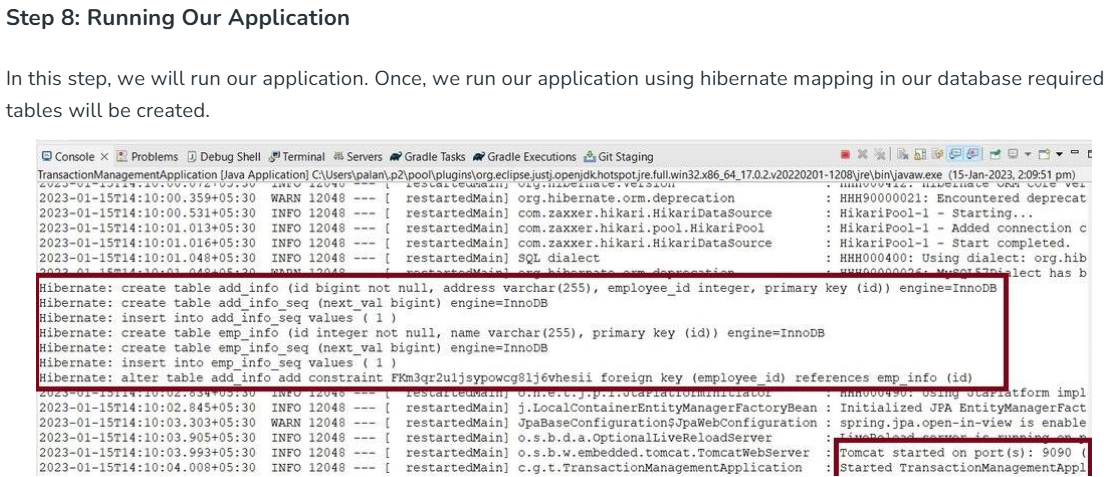


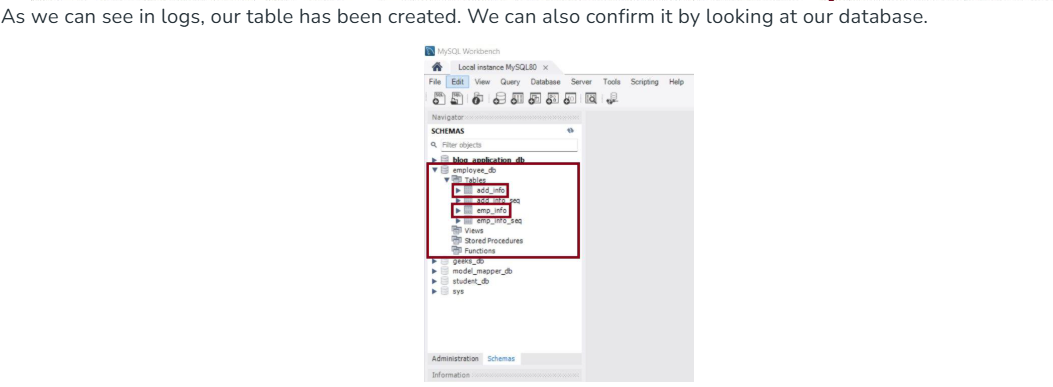


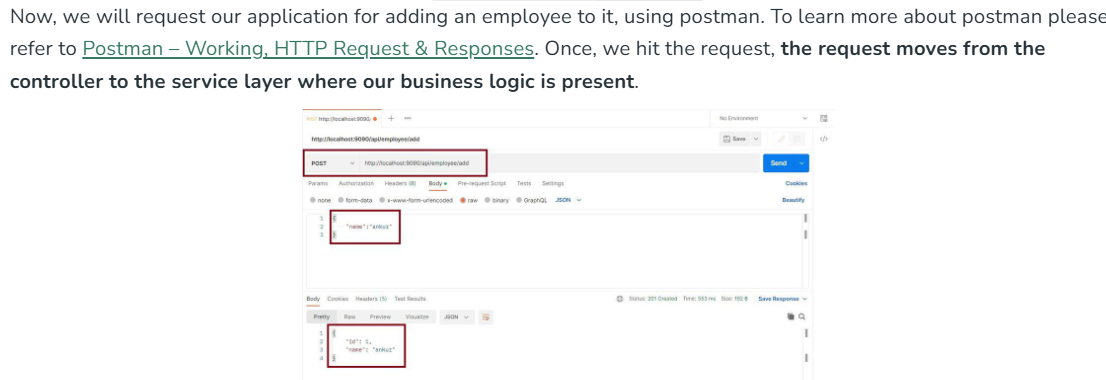


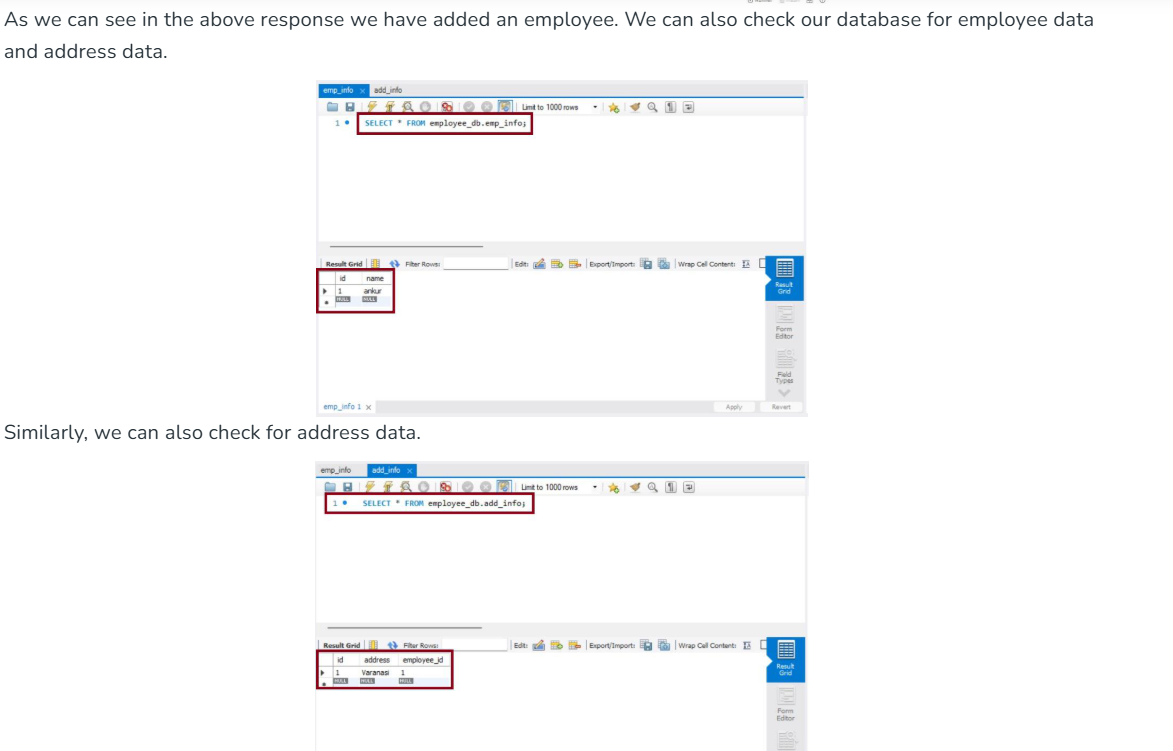






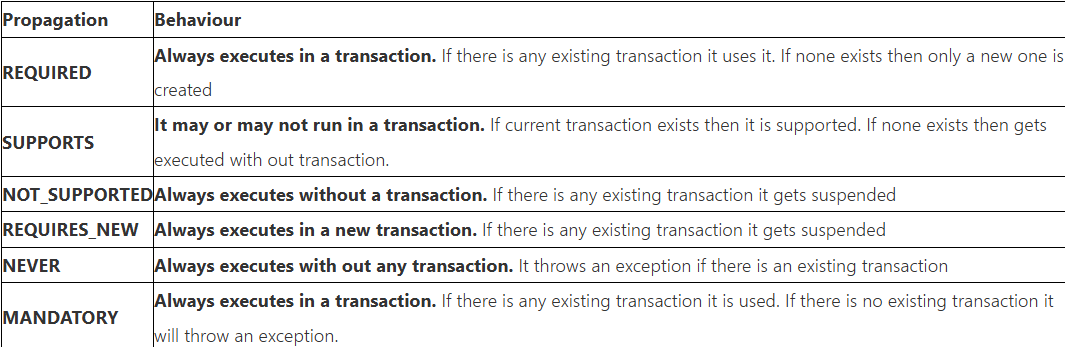




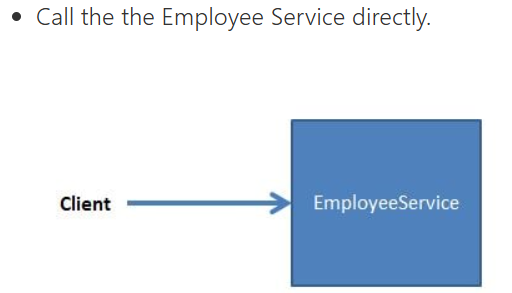


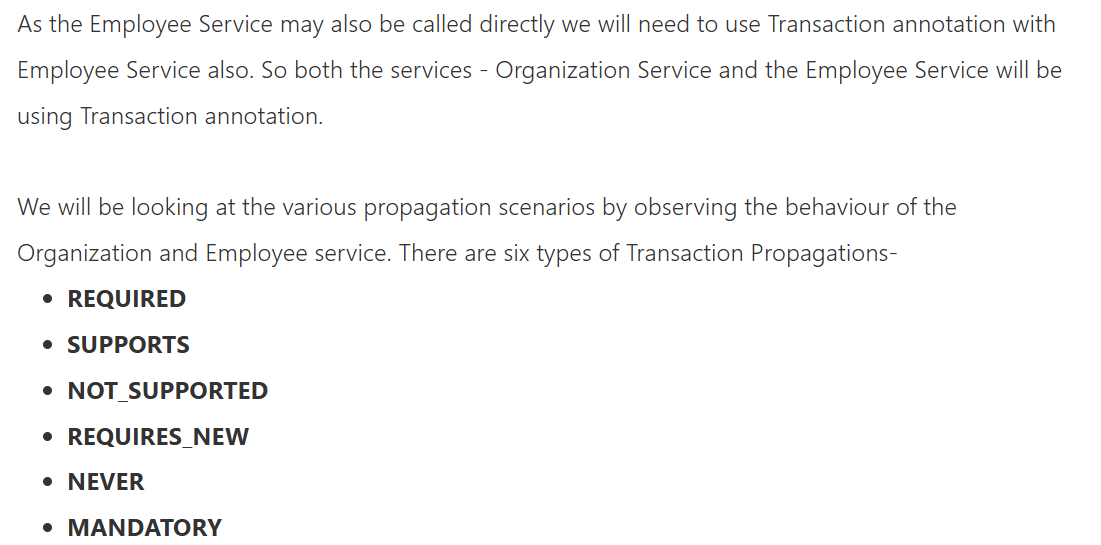
1. **Explain different types of propagation in transaction management?**

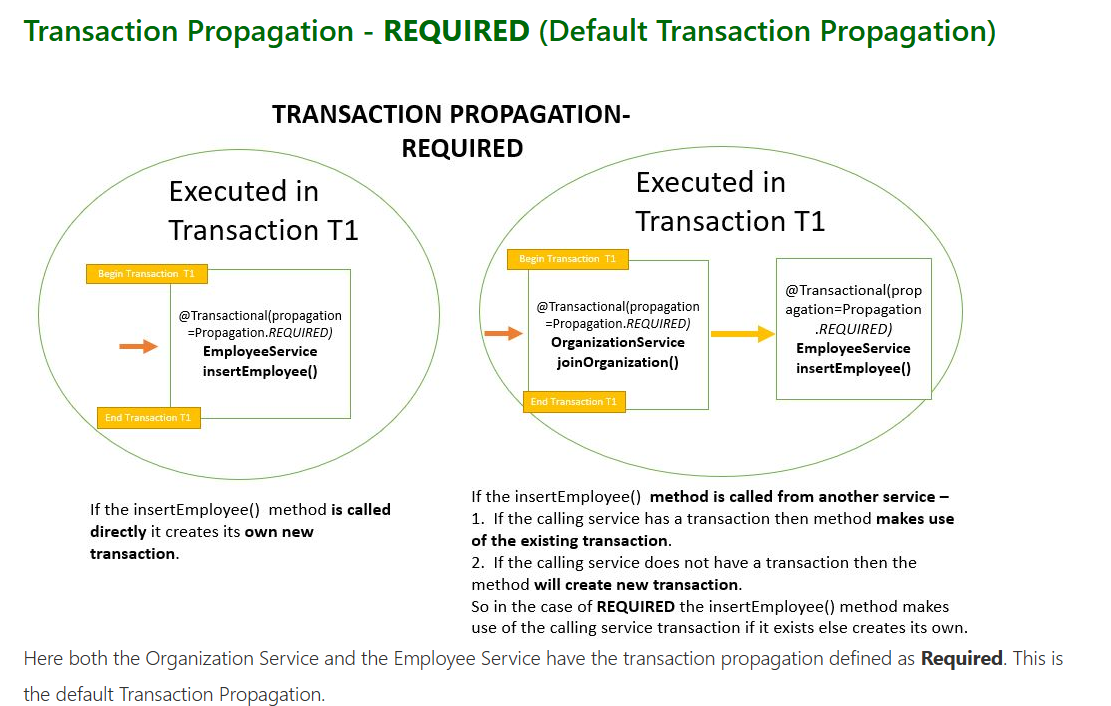
Transaction propagation determines how transactions interact when multiple transactional methods are executed. Spring provides different **propagation types** to define how transactions behave within nested calls.



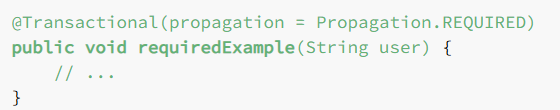


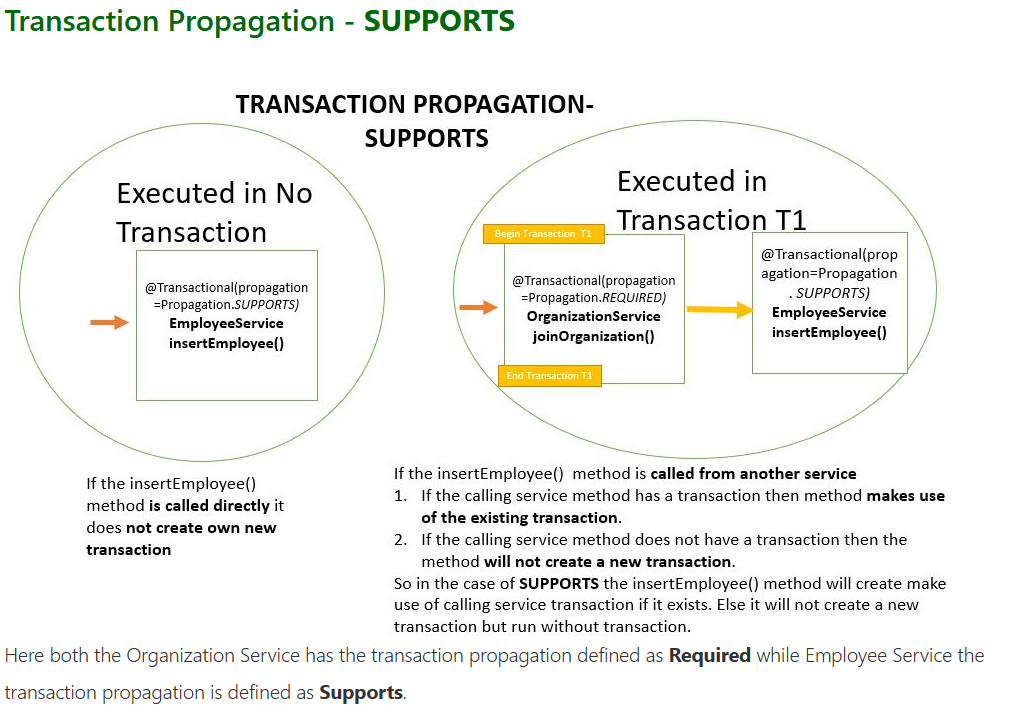


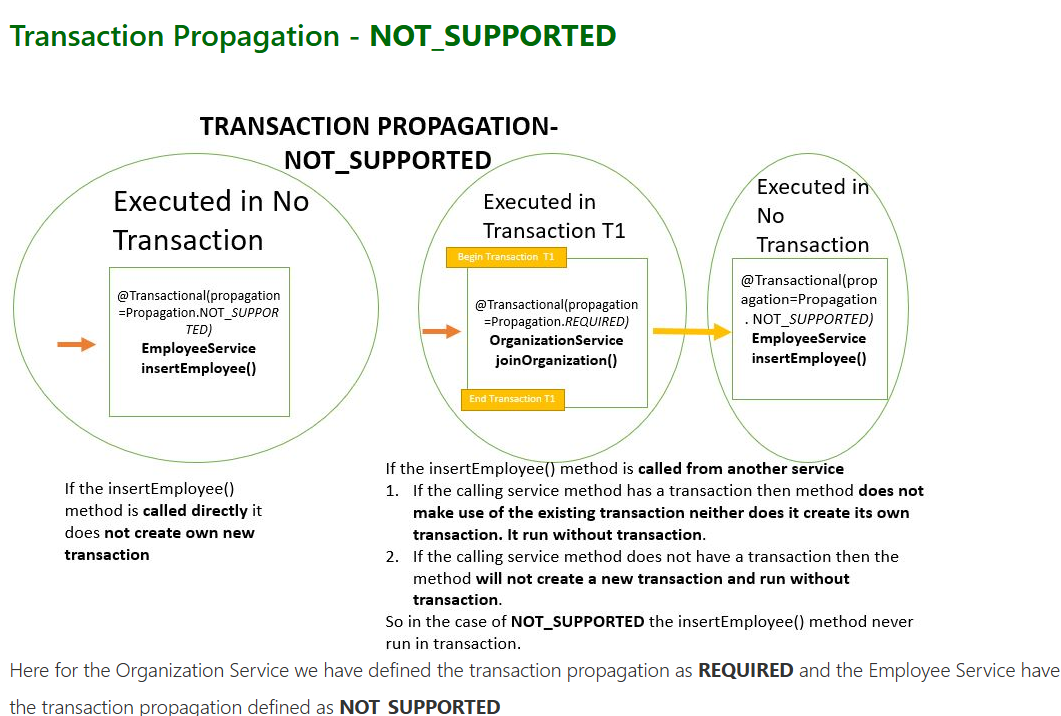


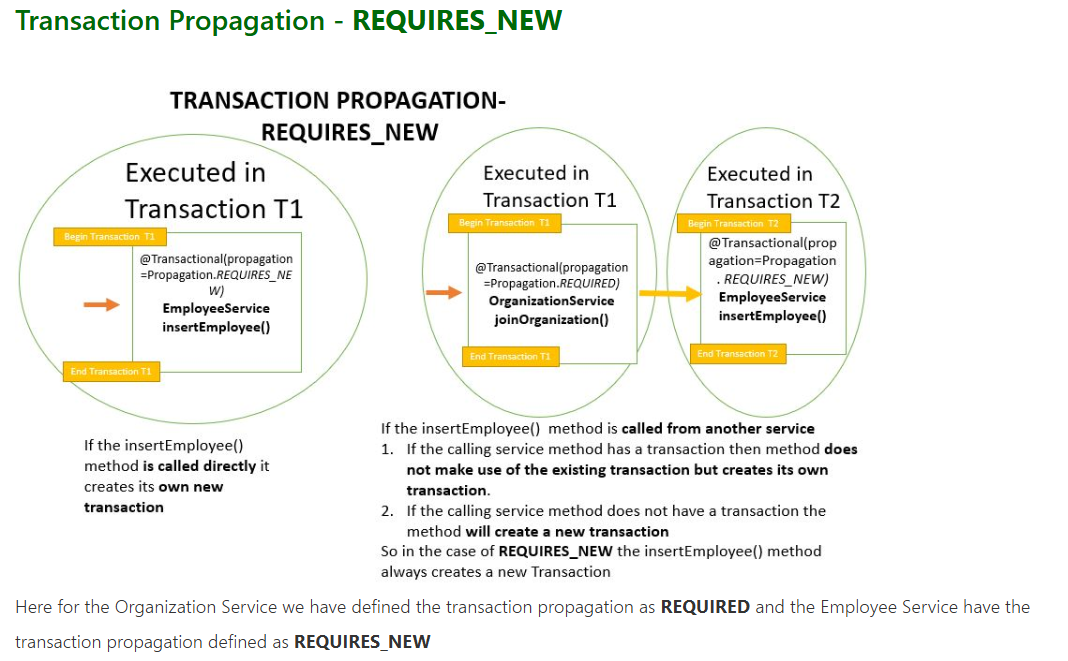


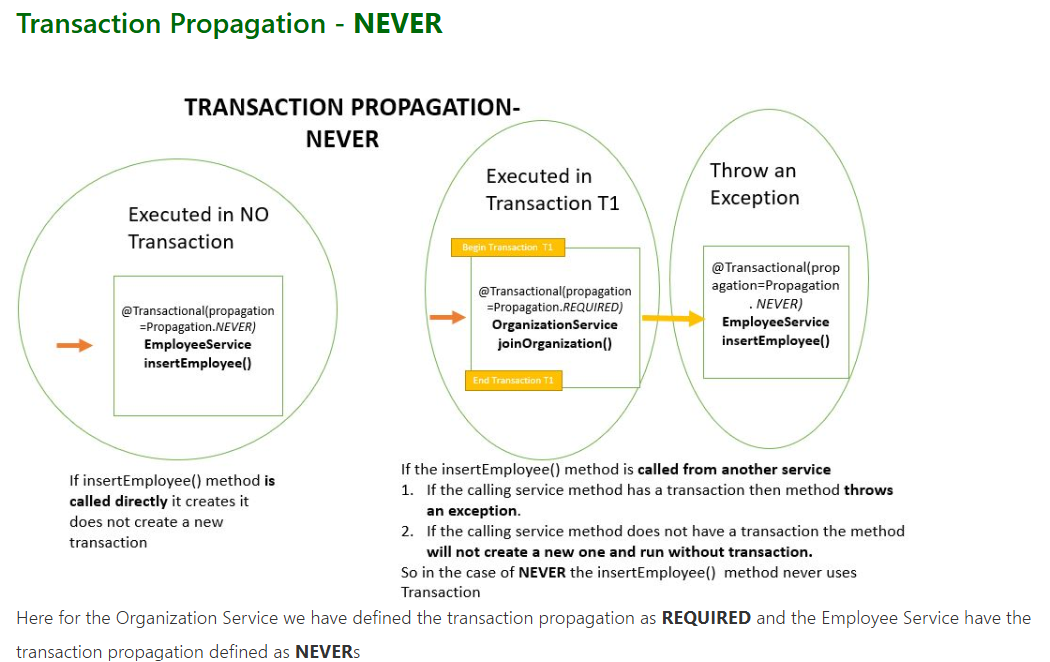
# Example

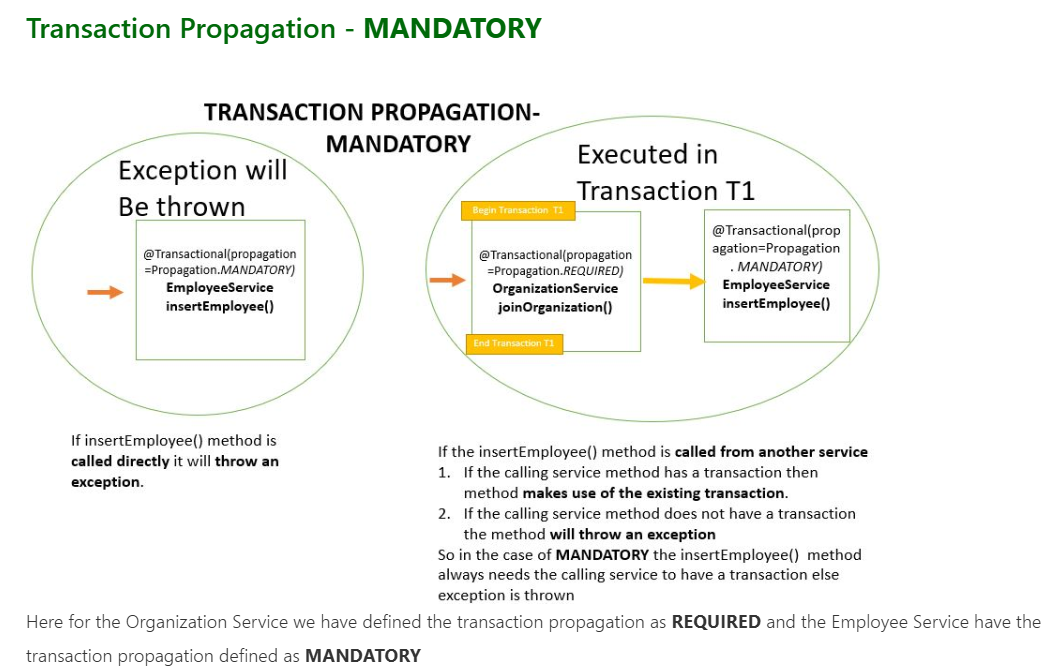






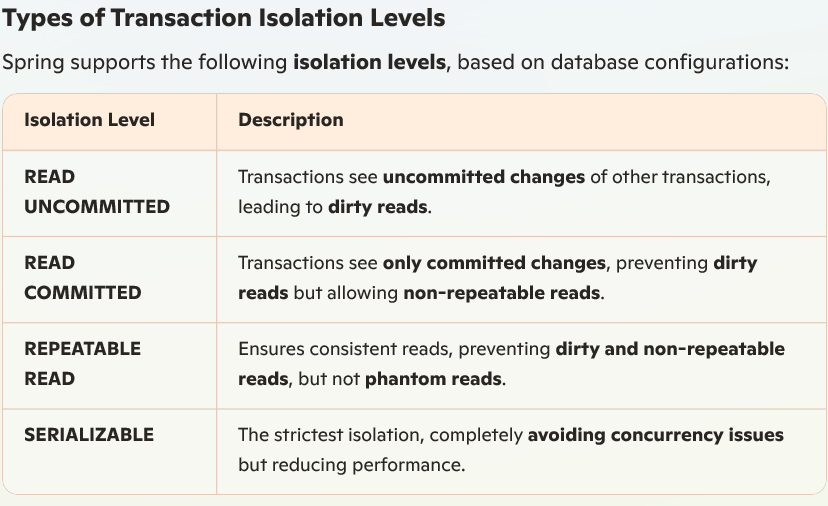






1. **Explain about different types of Transaction Isolation?**

**Transaction Isolation** controls **how and when** the changes made by one transaction become visible to other concurrent transactions. It's a key part of maintaining **data consistency**.



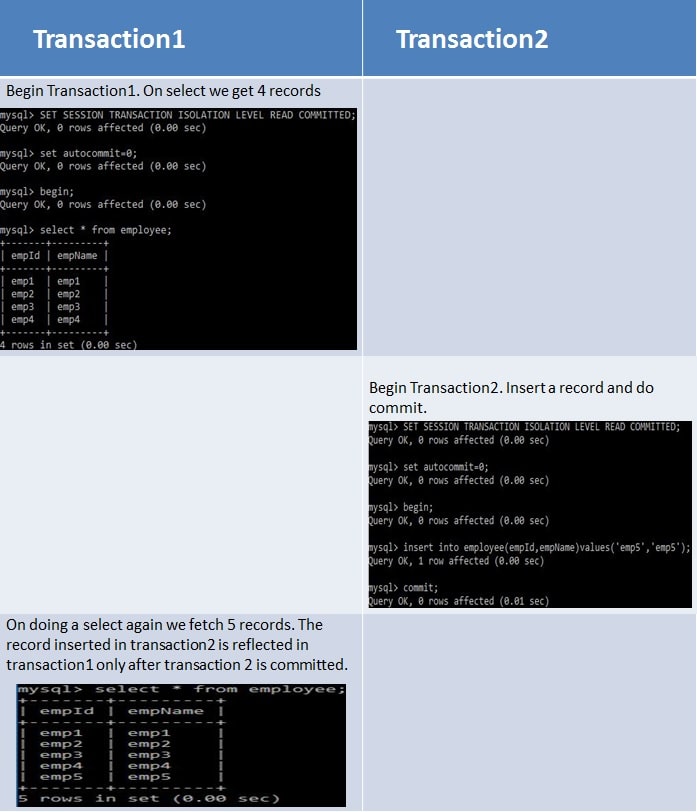
SERIALIZABLE  
If two transactions are executing concurrently then it is as if the transactions get executed serially i.e the first transaction gets committed only then the second transaction gets executed. This is **total isolation**. So a running transaction is never affected by other transactions. However this may cause issues as **performance will be low and deadlock might occur**.



REPEATABLE\_READ  
If two transactions are executing concurrently - **till the first transaction is committed the existing records cannot be changed by second transaction but new records can be added.** After the second transaction is committed, the new added records get reflected in first transaction which is still not committed. For MySQL the default isolation level is REPEATABLE\_READ.  
However the REPEATABLE READ isolation level behaves differently when using mysql. When using MYSQL we are not able to see the newly added records that are committed by the second transaction.

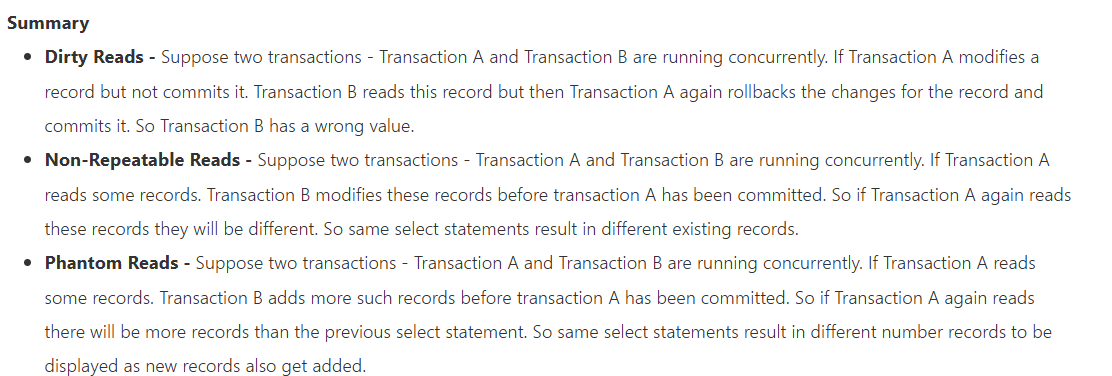


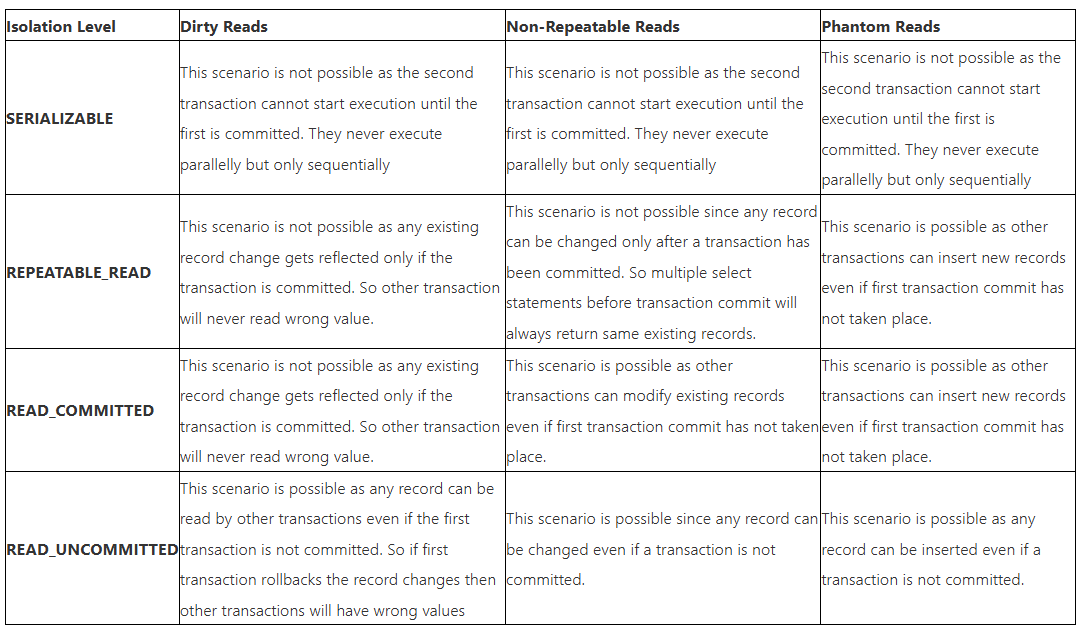
READ\_COMMITTED  
If two transactions are executing concurrently - **before the first transaction is committed the existing records can be changed as well as new records can be changed by second transaction.** After the second transaction is committed, the newly added and also updated records get reflected in first transaction which is still not committed.



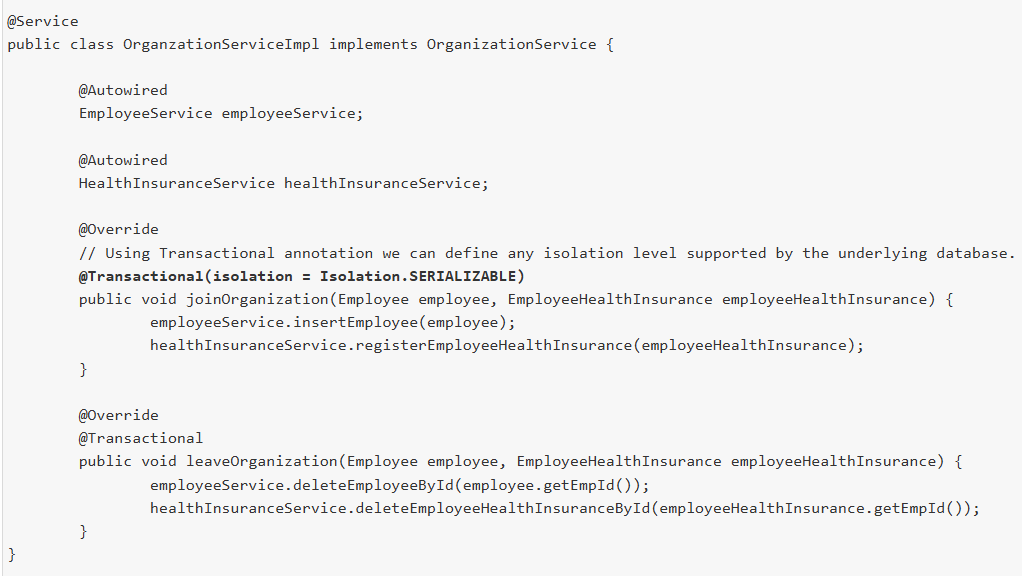
READ\_UNCOMMITTED  
If two transactions are executing concurrently - before the first transaction is committed the existing records can be changed as well as new records can be changed by second transaction. **Even if the second transaction is not committed the newly added and also updated records get reflected** in first transaction which is still not committed.





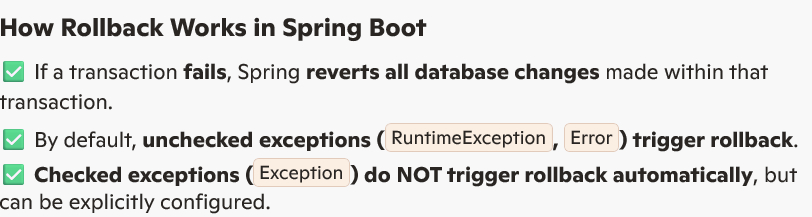


# Example:

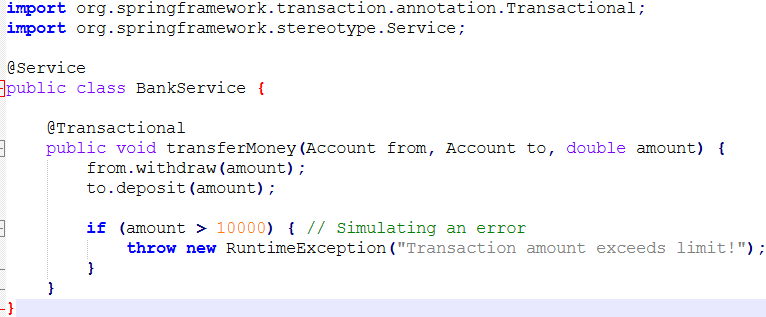


1. **Explain Rollback Mechanism in Spring Boot Transaction Management?**

Rollback ensures **data integrity** by undoing changes **when an error occurs** during a transaction. Spring Boot **automatically handles rollback** in transactions based on **exception types and configurations**

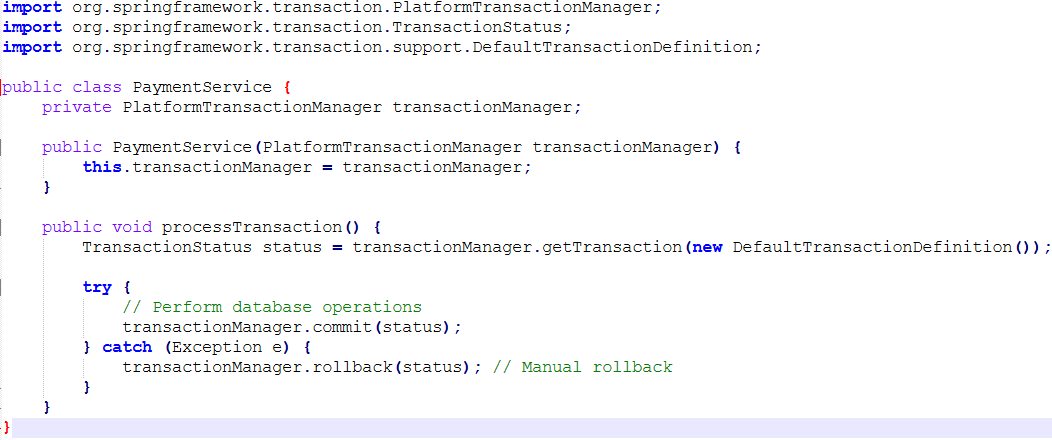






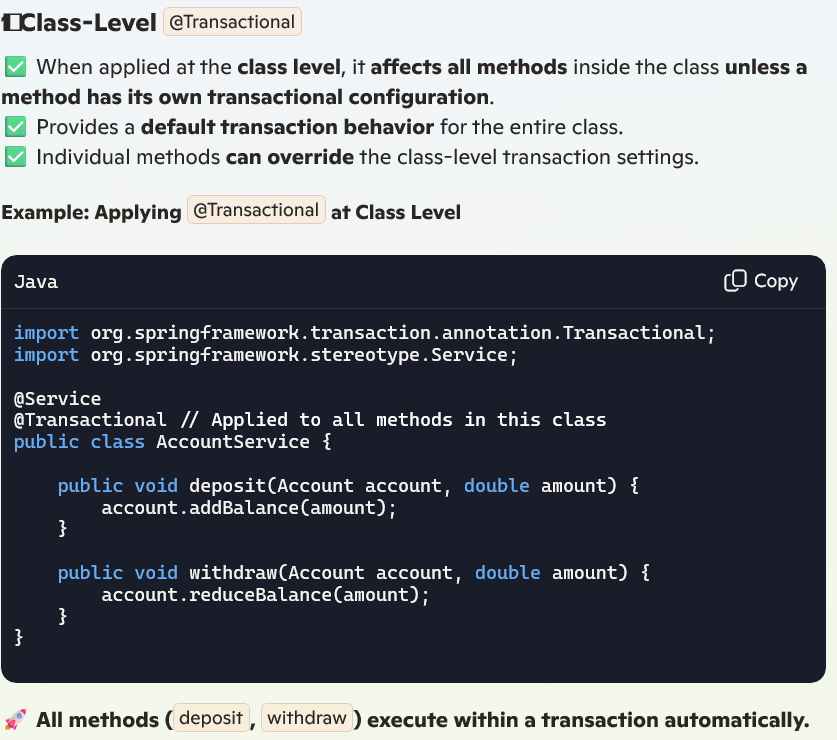


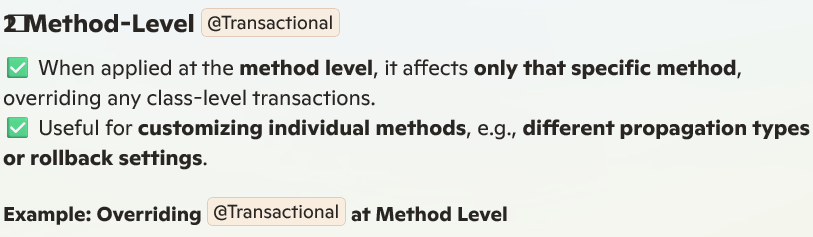


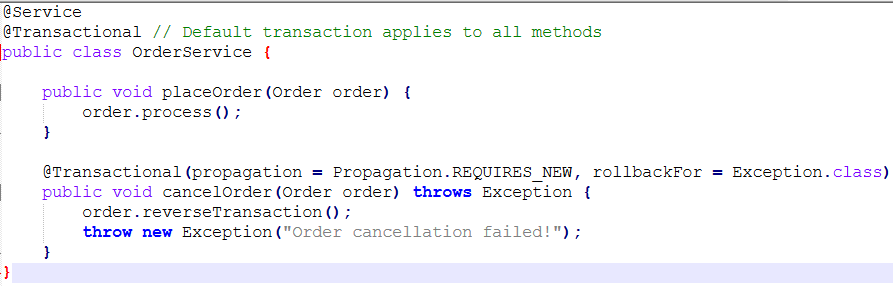


1. **Can we apply @Transactional in class level and method level?**

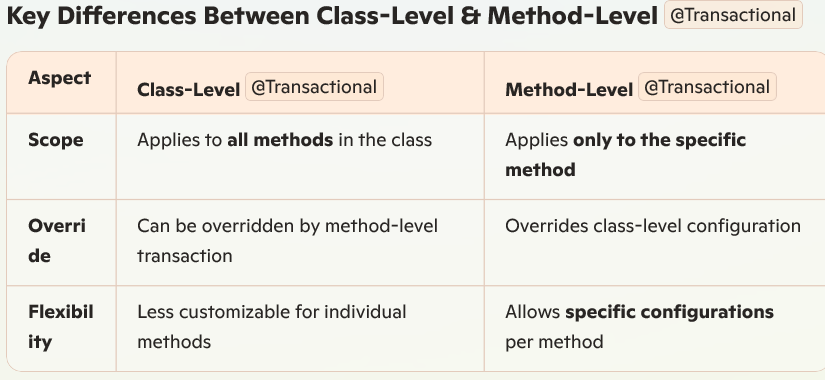
Yes, **@Transactional** can be applied at both the **class level** and **method level** in Spring.



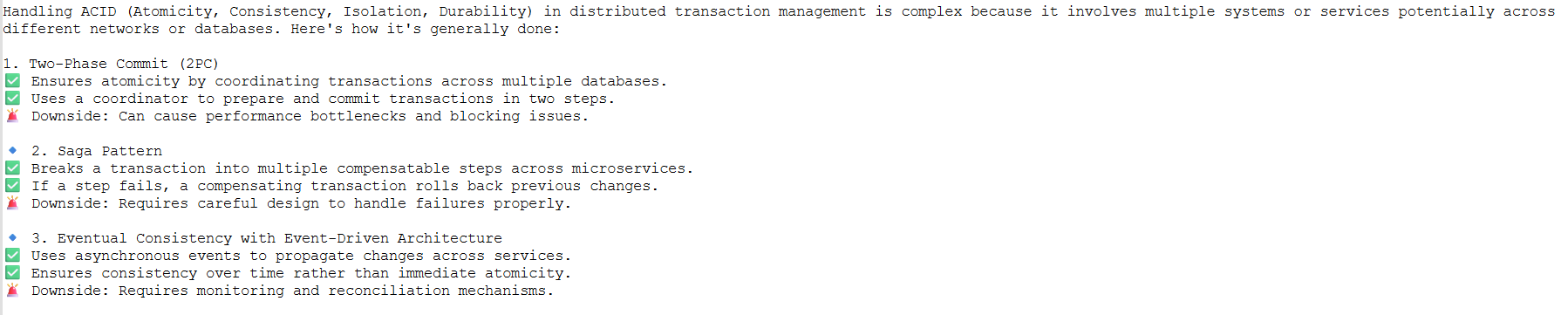




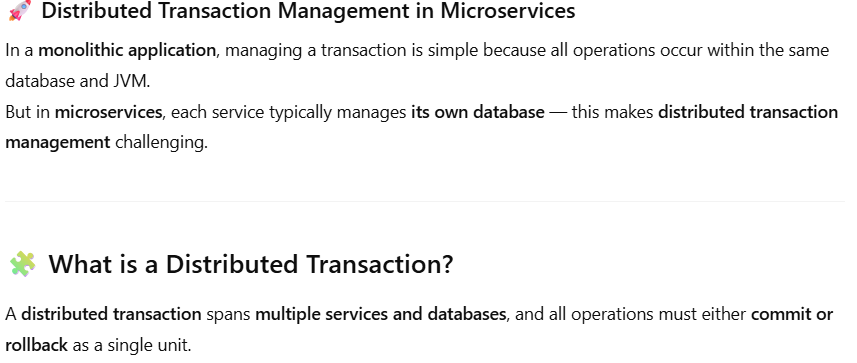


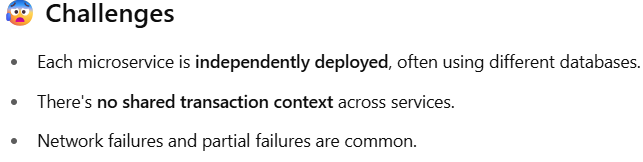


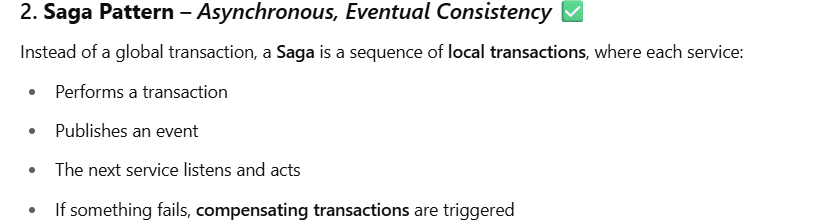
1. **How to handle distributed transaction management in spring boot micro services?**

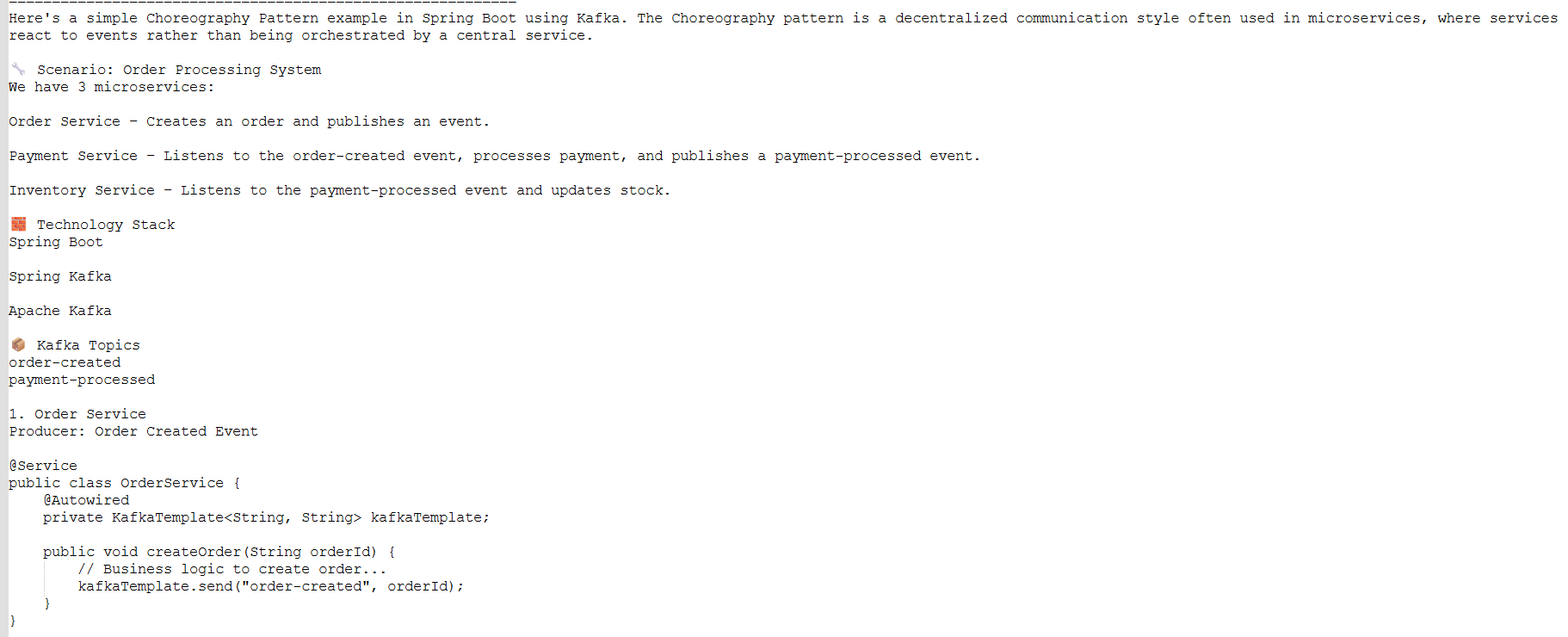


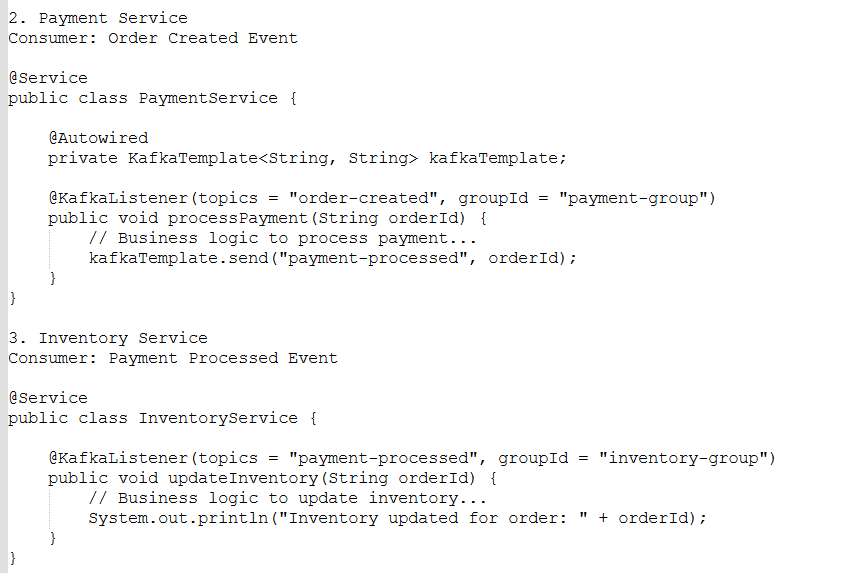
1. **Explain distributed transaction management in micro services?**





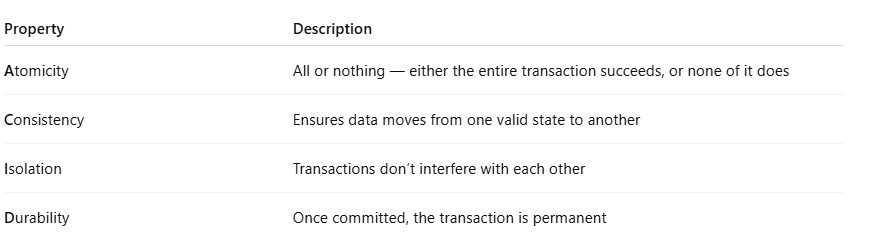


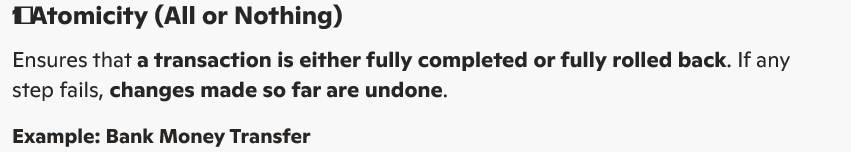




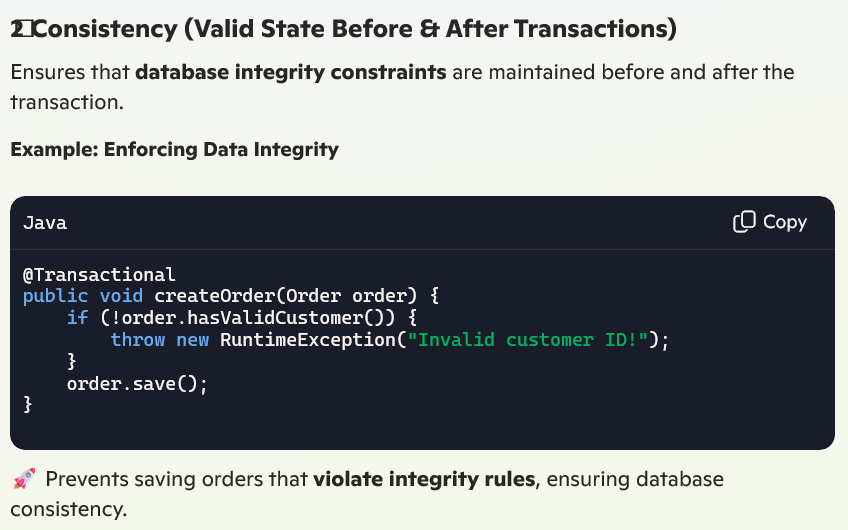


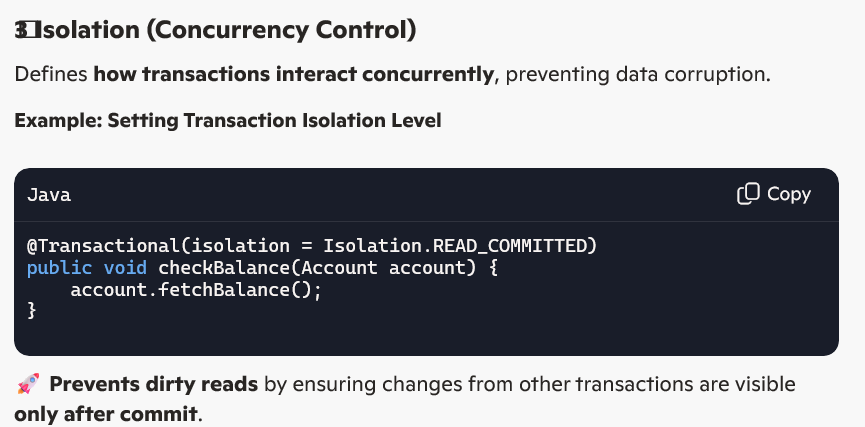
1. **Explain ACID properties?**

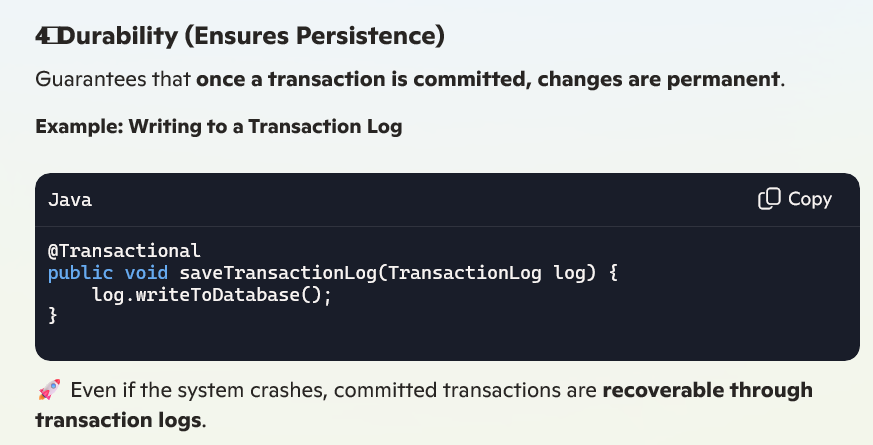




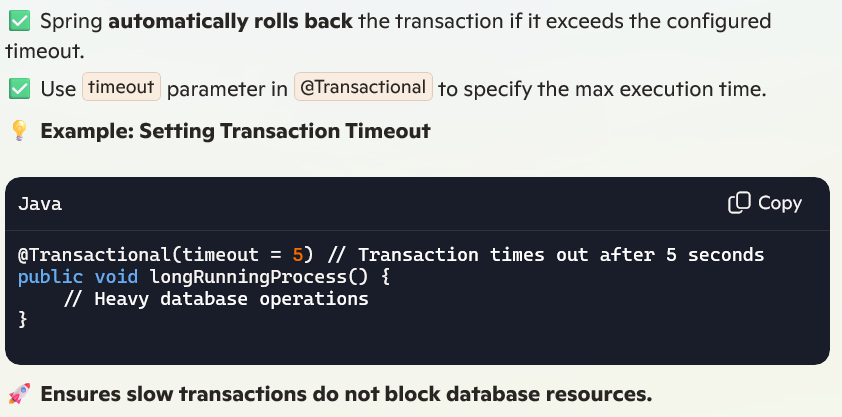




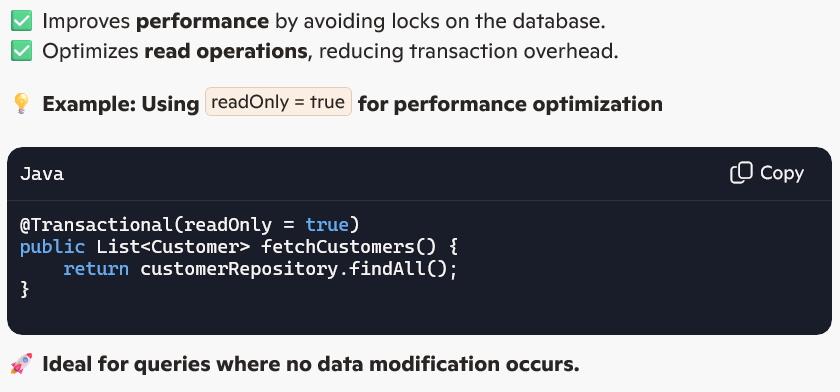




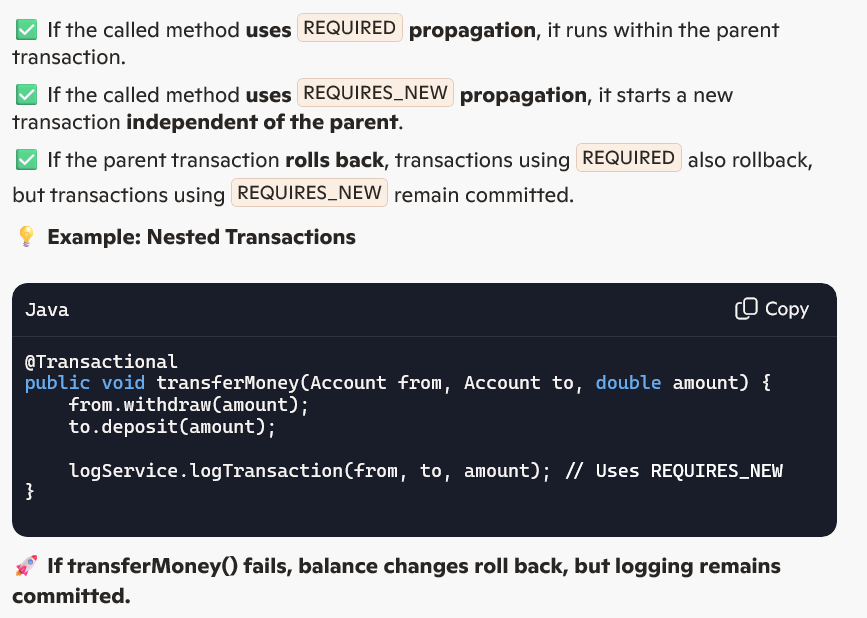
1. **What happens if a transaction times out in Spring?**



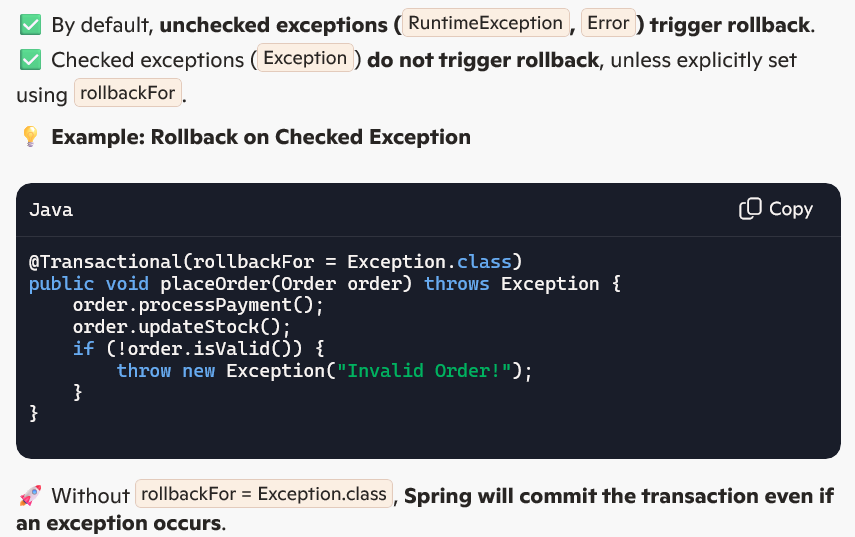
1. **What is the benefit of using @Transactional(readOnly = true)?**

****

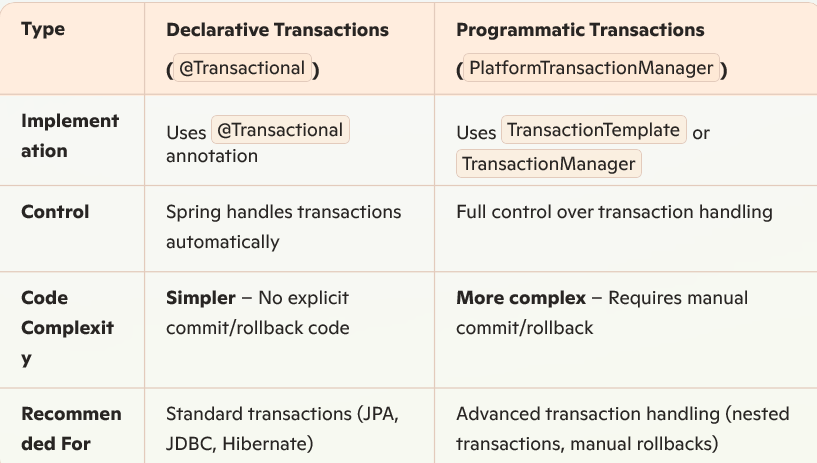
1. **What happens if you call a transactional method inside another transactional method?**

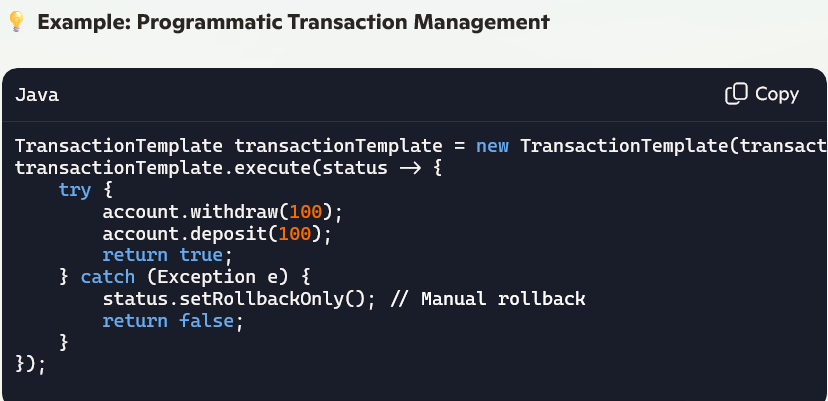
****

1. **How does Spring handle rollback in transactions?**

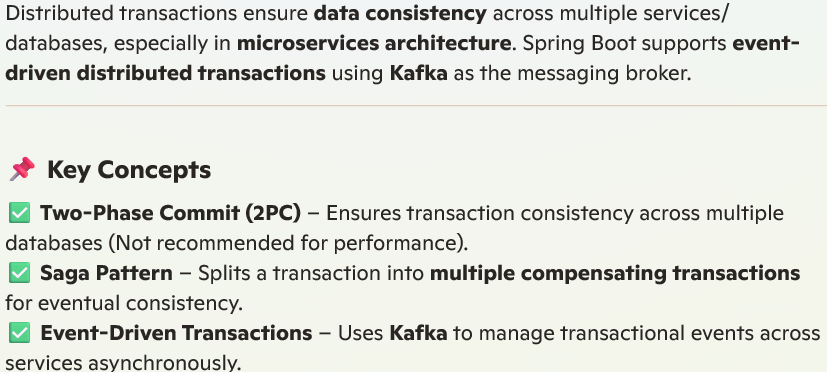
****

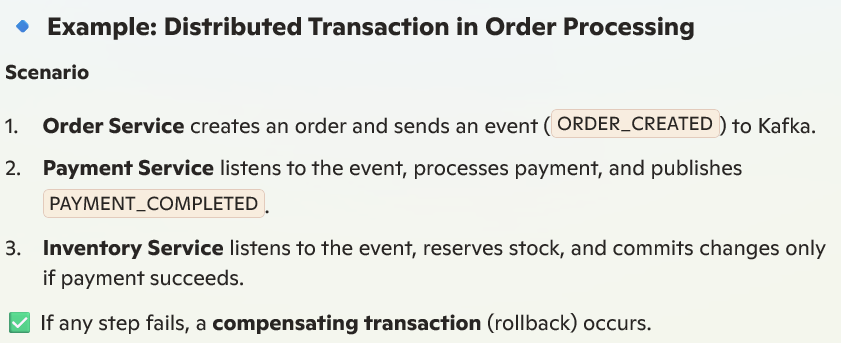
1. **What is the difference between Declarative and Programmatic Transaction Management?**

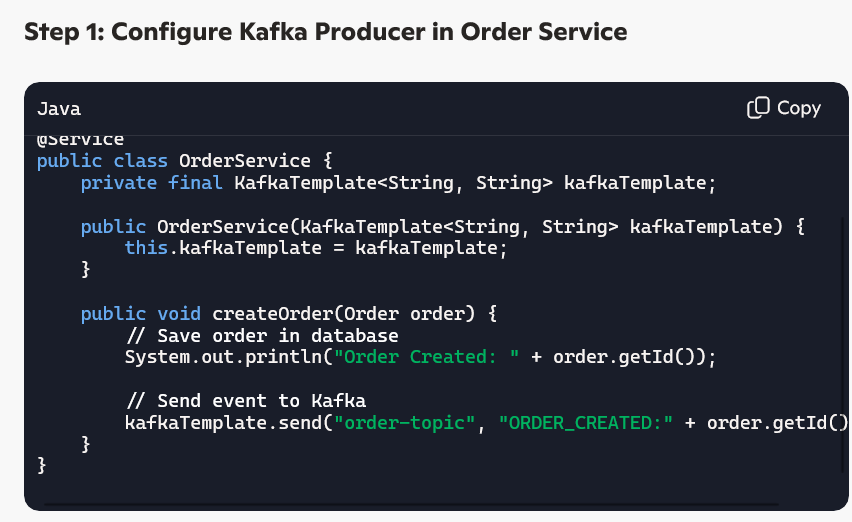
****

****

1. **Explain realtime Spring Boot Distributed Transactions with Kafka Example?**

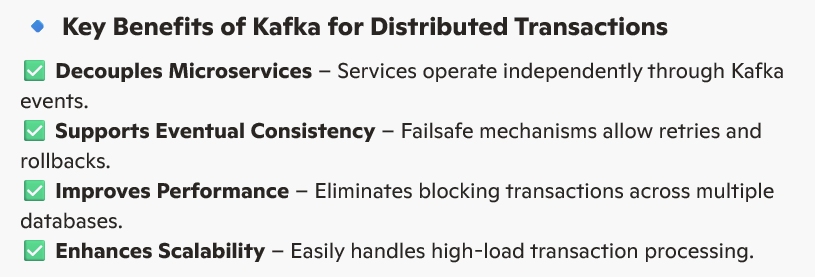
****

****

****

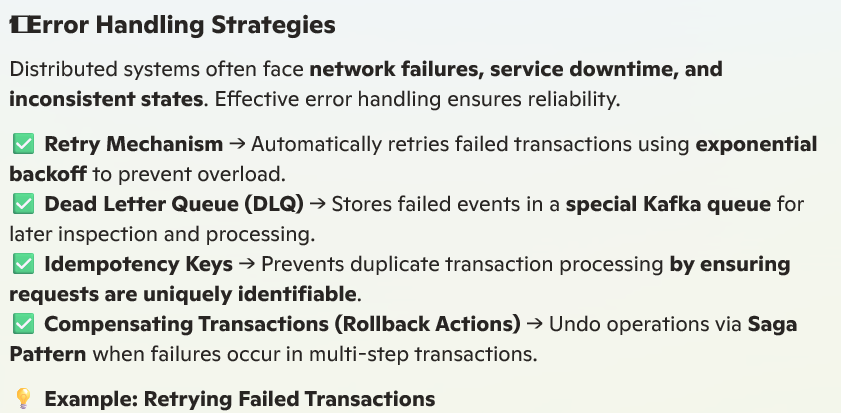
****

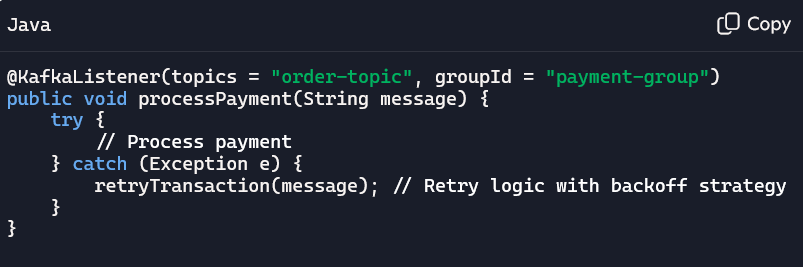
****

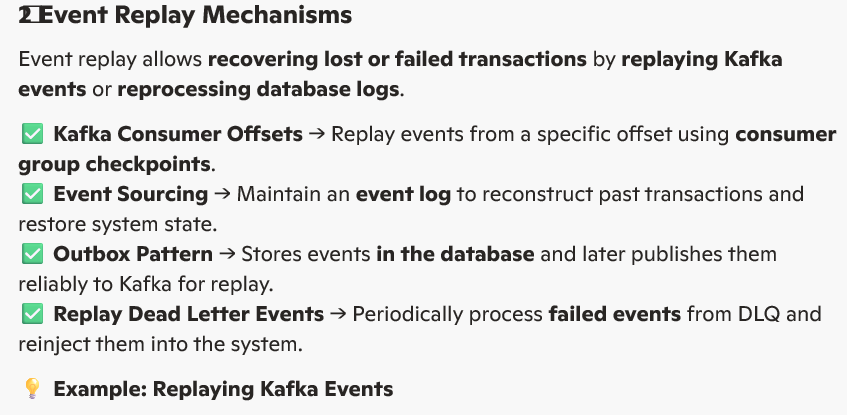
****

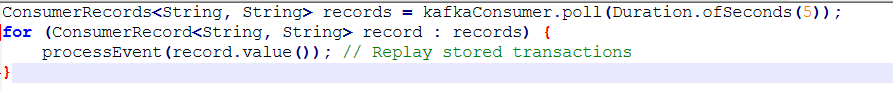
**16. Error handling strategies or event replay mechanisms in distributed transactions?**

Distributed transactions require robust error handling and event replay mechanisms to ensure data consistency and fault tolerance in microservices. Here are key techniques:



****

****

****